VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 Natalya Zarapina

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ANOTACE OBRAZU S POUŽITÍM HLUBOKÉHO UČENÍ
IMAGE ANNOTATION USING DEEP LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Natalya Zarapina
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Radim Burget, Ph.D.
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Studentka: Natalya Zarapina ***ID:*** 174249
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Anotace obrazu s použitím hlubokého učení

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s nástroji Keras a Python a načtěte předtrénovanou neuronovou síť pro rozpoznávání objektů z obrazu. Vytvořte si vlastní trénovací databázi s oblečením a neuronovou síť přetrénujte pro takto zadaný problém. Ověřte přesnost a dosažené výsledky vhodně zaneste do grafů či tabulek.

DOPORUČENÁ LITERATURA:

[1] Weigel, Van B. Deep Learning for a Digital Age: Technology's Untapped Potential To Enrich Higher Education. Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, 2002.

[2] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12.Oct (2011): 2825-2830.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: doc. Ing. Radim Burget, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a programu typu klient-server pro klasifikaci a lokalizaci určitých elementů vyskytujících se na předložených obrazcích. Program načítá sadu obrázků a s využitím hlubokého učeni zejména hluboké konvoluční neuronové sítě provádí klasifikaci. V první části práce je popsaná architektura, základní principy fungování konvolučních sítí a také vybrané klasifikační algoritmy strojového učení. Druhá část obsahuje samotný popis navrženého programu.

KLÍČOVÁ SLOVA

anotace, hluboké učení, Keras, konvoluční neuronová síť, neuron, obrázek, Python, strojové učení, Theano, umělá neuronová síť

ABSTRACT

This semester thesis describes the design and implementation of the client-server program for classification and localization of certain elements which are present in provided images. This program loads a set of images and use deep learning, especially deep convolution neural network perform a classification. First part describes the architecture, basic principles of operations in convolution network and chosen machine learning algorithms for classification. Second part contains a description of created program.

KEYWORDS

Machine learning, deep learning, neuron, artificial neural network, convolutional neural network, Python, Theano, Keras, annotation, image

ZARAPINA, Natalya *Anotace obrazu s použitím hlubokého učení* : bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2017. 43 s. Vedoucí práce byl doc. Ing. Radim Burget, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Anotace obrazu s použitím hlubokého učení “ jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky

PODĚKOVÁNÍ

Ráda bych poděkovala vedoucímu bakalářské práce panu doc. Ing. Radimovi Burgetovi, Ph.D. za jeho nezbytné rady, podnětné návrhy, trpělivost, ochotu a také za čas, který mi věnoval při řešení dané problematiky.

Brno ....

podpis autorky

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autorky

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Člověk je pořád nucen rozpoznávat, rozhodovat a učit se. Pro nás je zcela přirozené poznávat různé jak známé, tak i neznámé objekty bez ohledu na jejich velikost, polohu, úhel pohledu nebo okolí, ve kterém se objekt nachází. Když se třeba díváme na nějaký obrázek, snadně a podvědomě ho můžeme popsat. Ale pro počítač je obrázek jen sada pixelů a tento úkol pro něj není vůbec jednoduchý. Touto problematikou se zabývá podoblast umělé inteligence, která se nazývá *strojové učení.*

Strojové učení – často také *machine learning* vzniklo současně se vznikem prvních počítačů a po dobu posledních 70 let je plně se rozvíjející disciplínou. Tato disciplína se zabývá algoritmy a technikami, které se učí z poskytnutých dat, přitom učení v tomto smyslu znamená, že se programy budou chovat požadovaným způsobem, i když k němu nebyly přímo naprogramovány. K základním typům úloh strojového učení patří klasifikace, regrese a hledání skupin.

Hluboké učení je sadou algoritmů strojového učení. Prudký nárůst výkonu GPU (grafický procesor, který zajišťuje rychlé grafické výpočty) a zvýšení množství dostupných dat mělo za následek rapidní rozmach ve využití hlubokého učení. Hluboké učení se uskutečňuje v několika úrovních. Podrobně analyzovat celý obrázek nejde zatím ani vyspělým moderním technologiím úplně bez problémů, proto je třeba nejprve identifikovat místa, o která se zajímáme. Každá nižší úroveň má představu o různých charakteristických rysech na daném obrázku, jako jsou hrany nebo obrysy, a následně je předává na další vyšší úrovně. V nejvyšších úrovních už tyto příznaky budou odpovídat složitějším reprezentacím na obrázku. Hluboké učení umožňuje pomocí tohoto procesu vytvořit hierarchickou strukturu. Nejčastějším případem použití hlubokého učení je aplikace v neuronových sítích.

Teoretická část této bakalářské práce je zaměřená na popis vybraných klasifikačních algoritmů strojového učení, kde podrobnější popis patří speciálnímu druhu vícevrstvových konvolučních neuronových sítí, pomocí kterých je realizována praktická část. V praktické části se práce zabývá extrakci příznaků z obrazu s použitím hlubokého učení. Hluboká neuronová síť byla upravena tak, že vyčítala hodnoty z předposlední vrstvy modelu. Tato vrstva totiž obsahuje informace o nízkoúrovňových příznacích z obrazu a může být využita k adaptaci na konkrétní případ, např. o vyskytujících se elementech jako jsou části oblečení apod. Na základě těchto dat by se dalo určit, co za oblečení se na daných obrázcích vyskytuje. Extrahované příznaky byly pak také využity pro ověření přesnosti jednotlivých klasifikačních algoritmů strojového učení v programu Rapidminer, který slouží pro analýzu dat. Na základě těchto výsledků bude natrénován vlastní model hluboké neuronové sítě, schopný rozpoznávat určité druhy oblečení. Bude také vytvořen program pro klasifikaci obrazu, který bude z důvodu snížení hardwarových požadavků rozdělen na

část klienta a serveru, aby se omezila potřeba opakovaně načítat neuronovou sít pro každý obrázek. Toto rozdělení nám také teoreticky umožní umístit serverovou část na výkonnější platformu a klienta pak spouštět z jakéhokoli počítače v lokální síti. Serverová část vytvořeného řešení totiž obstarává tu nejnáročnější část celého procesu analýzy.

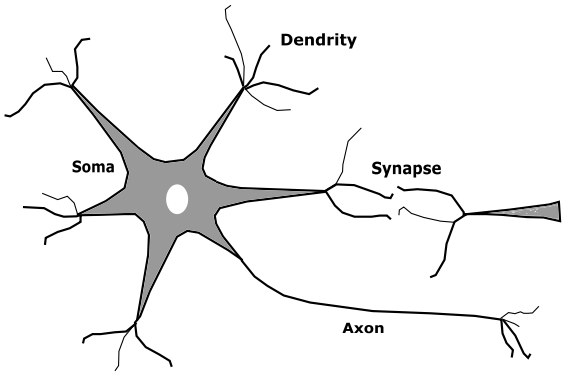
Hlavním přínosem této bakalářské práce je navrhnuta aplikace, která umožňuje z prostředí jazyka Java adaptabilní rozpoznání obsahu a také lokalizaci různých druhů oblečení z libovolných obrazů pomocí hluboké konvoluční neuronové sítě. Výsledná aplikace bude díky rozdělení na část klienta a serveru dostatečně rychlá i pro praktické nasazení v oděvním průmyslu. Pro jednoduchost při využití neuronových sítí bude také vytvořen instalátor pro prostředí operačních systémů Microsoft Windows, který obstará instalaci všech potřebných programů pro správnou funkci tohoto systému.

1 KONVOLUČNÍ NEURONOVÁ SÍŤ

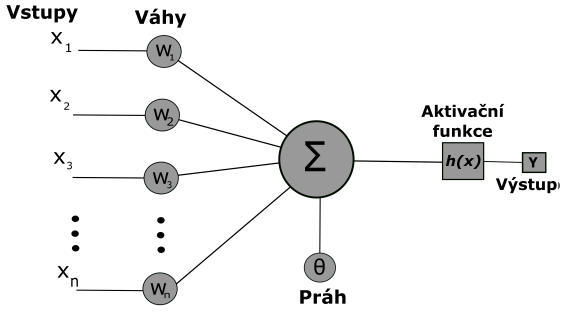
Konvoluční neuronová síť je speciálním druhem vícevrstvových umělých neuronových sítí. Poprvé byly tyto sítě použity pro rozpoznávání ručně psaných číslic v roce 1980. Největší pozornost konvoluční sítě upoutaly ale až v roce 2012 po soutěži *ImageNet Large Scale Visual Recognition Challenge* (zaměřená na rozpoznání obrazu). Kde Krizhevsky a kolektiv, použili konvoluční neuronové sítě, a vyhráli soutěž s chybovostí pouze 15,3%, čímž prokázali jejich vysokou výkonnost. Od té doby došlo ke zvýšení zájmu o konvoluční neuronové sítě, a nyní jsou stále hodně používané pro klasifikaci obrazů do tříd nebo také rozpoznání objektů v obraze.

1.1 Architektura sítě

Vývoj umělých neuronových sítí je inspirován biologií, hlavně strukturou nervové soustavy. Umělé neuronové sítě byly založené na principu fungování neuronu, který je schopen přijímat, zpracovávat a přenášet elektrochemické signály podél nervových drah, které tvoří komunikační systém mozku. Neurony se skládají ze ***somy*** (těla neuronu), ***axonu*** (dlouhého výběžku) a ***dendritů*** (krátkých výběžků). Místo spojení mezi neurony se nazývá ***synapse***, které umožňují předávání signálů [1]. Na obr. 1.1 je ukázaná struktura typického biologického neuronu.



Obr. 1.1: Biologický neuron



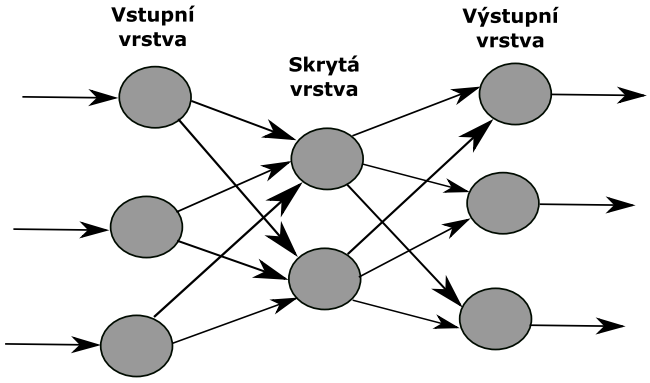
Obr. 1.2: Umělý neuron

Pojem umělé neuronové sítě byl navržen již v roce 1943 W. McCullochem a W. Pittsem. Konkretně jimi byl navržen umělý model neuronu, kterému se říká ***perceptron*** (obr. 1.2). Signály přenášející se podél axonů odpovídají n vstupům $(x_1, \ldots, x_n)$, každý vstupní signál se vynásobí odpovídající váhou $(w_1, \ldots, w_n)$. Hodnoty vah se nastavují a upravují v průběhu učení. Aktivační hodnota nebo také vnitřní potenciál neuronu je součet všech vážených vstupů. Umělý neuron má také práh θ, který stanoví prahovou hodnotu jeho aktivace, při překonání této hodnoty se neuron aktivuje

a pošle signál na výstup Y ve formě aktivační funkce $F(x)$ [2]. Fungování neuronu pak lze definovat tímto matematickým vztahem:

$$Y = F(\sum_{i=1}^{n} x_i w_i + \theta). \tag{1.1}$$

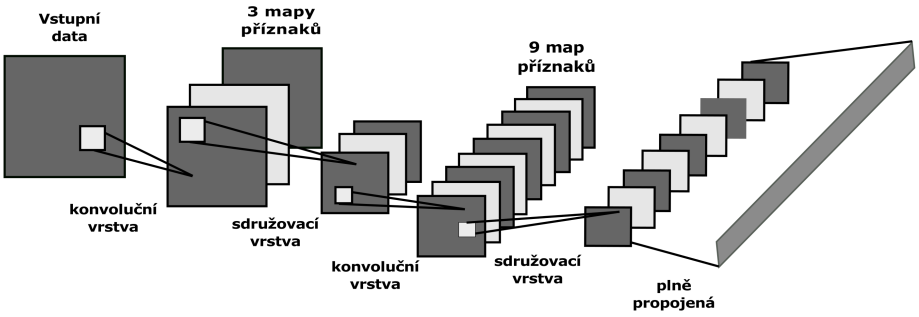
Samotný neuron není ale schopný provést jakoukoli složitější funkci. Neuron je pouze základní výpočetní jednotkou umělých neuronových sítí. Jak již bylo zmíněno, umělé neuronové sítě se skládají z vrstev. Vrstva je utvořená souhrnem neuronů, jejich vstupy jsou spojené s výstupy neuronů předcházející vrstvy, rovněž výstupy této vrstvy budou vstupy vrstvy následující. První vrstva, které se říká vstupní vrstva, přijímá a převádí vstupní data na vstupy neuronu další vrstvy. Poslední vrstva se nazývá výstupní vrstvou, ona určuje výstup neuronové sítě. Vrstvy, které se nachází mezi vstupní a výstupní vrstvou, se nazývají skryté vrstvy, protože ani jejich vstupy nebo výstupy, které se v nich nachází, nezasahují do vnějšího prostředí [3]. Počet skrytých vrstev roste se složitosti funkce, kterou má síť splnit. Existuje hodně architektur umělých neuronových sítí, první a nejjednodušší z nich jsou dopředné sítě (viz obr. 1.3).



Obr. 1.3: Dopředná neuronová síť

Dopředné jsou protože při průchodu dat od vstupní do výstupní vrstvy, neurony můžou být spojené jen se vstupy neuronů následující vrstvy, tudíž nevzniknou žádné smyčky a data postupují pouze dál až do konce. Jedním z představitelů dopředných neuronových sítí jsou sítě konvoluční.

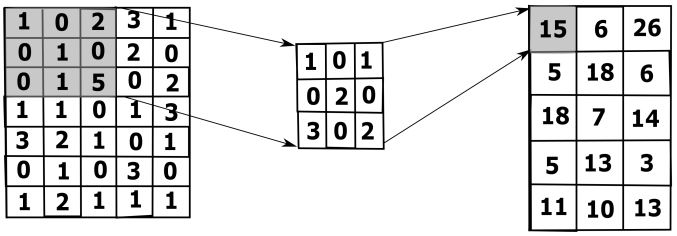
Hlavní myšlenka těchto sítí spočívá ve střídání konvolučních (kap. 1.1.1), sdružovacích (kap. 1.1.2) a plně propojených (kap. 1.1.3) výstupních vrstev. Příklad architektury konvoluční neuronové sítě je zobrazen na obr. 1.4



Obr. 1.4: Architektura konvoluční neuronové sítě

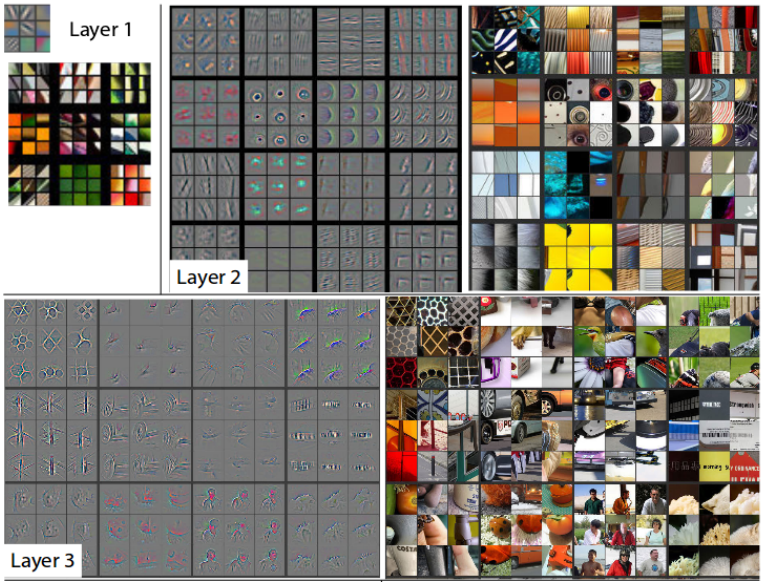
1.1.1 Konvoluční vrstva

Vstupem konvoluční neuronové sítě je například obrázek. Vstupní vrstvu nejčastěji následuje vrstva konvoluční. Matematická operace aplikující se nad obrázkem se nazývá konvoluce, ta je základem konvoluční vrstvy. Na obr. 1.5 je ukázán princip konvoluce. Vrstva se skládá z filtrů, kterým se říká ***konvoluční jádra*** (v odborné literatuře také *kernel*). Velikost těchto jader se definuje při vytváření vrstvy. Konvoluční jádro projde krok po kroku celým obrázkem. V každém kroku se původní hodnoty vynásobí příslušnými váhovými koeficienty konvolučního jádra, výsledky se sečtou a zapíšou do výsledné matice. Hodnota kroku určuje o kolik pixelů se jádro během tohoto procesu posune. Konvoluční vrstva vždy obsahuje těchto jader více. Jádra odpovídají určitým příznakům na obrázku, například jedno jádro bude



Obr. 1.5: Proces konvoluce

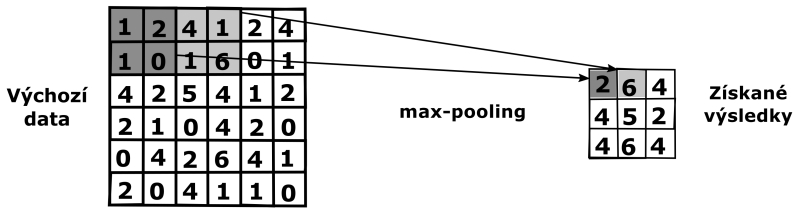
detekovat horizontální hrany, další jádro pak vertikální. Výstupem konvolučních vrstev jsou ***mapy příznaků*** (viz obr 1.6), které odpovídají elementům detekovaným jednotlivými konvolučními jádry, jejich počet odpovídá počtu aplikovaných jader [5].



Obr. 1.6: Mapy příznaků na různých vrstvách. Převzato z [6]

1.1.2 Sdružovací vrstva

Často také pooling vrstva, obvykle se vkládá po konvoluční vrstvě, jejím úkolem je slučování okolních pixelů do jednoho, čímž se zajistí snižování rozměrů příznakových map vytvořených konvolučními vrstvami. Nejčastěji se pro tuto vrstvu požívá metoda max-pooling. Funguje to skoro stejně, jako u konvoluční vrstvy. Přes celou vytvořenou mapu příznaků postupuje jádro, které z matice hodnot vybere maximální výstupní hodnotu. Oproti konvoluční vrstvě je rozdíl v tom, že jádro se nepřekrývá s předešlou pozicí. Princip fungování sdružovací vrstvy je zobrazen na obr. 1.7. Hlavním plusem pooling vrstvy je vyloučení hodnot, které nezahrnují maxima, což má tedy za následek zrychlení výpočtu a sníženi potřebných nároků na dostupnou paměť [7].



Obr. 1.7: Proces max-poolingu

1.1.3 Plně propojená vrstva

Plně propojená vrstva představuje výstupní vrstvu konvolučních sítí. Počet neuronů v této vrstvě odpovídá přesnému počtu klasifikačních tříd. Výstupy z příznakových map poslední konvoluční nebo sdružovací vrstvy jsou přetransformovány do jednorozměrného vektoru. Tento vektor je pak převeden jako vstup na každý neuron této výstupní vrstvy, kde jsou vypočítané skalární součiny vstupních hodnot.

Dropout technika se používá především u plně propojených vrstev. Zásadní funkcí této metody je zabránit přeučení sítě. Dropout technika upravuje počet aktivních neuronů ve vrstvě. Při učení se pro každý cyklus deaktivuje stanovený počet neuronů. Počet deaktivovaných neuronů je koeficient v intervalu 0 až 1. Obvykle se používá koeficient 0,5. Znamená to, že v každém cyklu se náhodně vynuluje polovina neuronů, přitom druhá polovina neuronů zůstává aktivní, tj. každý cyklus trénovacího procesu budou natrénované různé neurony, ale na konci se výsledky zprůměrují.

1.2 Aktivační funkce

Aktivační funkce se aplikuje pro aktivaci neuronu na jeho výstup. Aktivační funkcí může být libovolná matematická funkce, ale měla by být nelineární. Nelinearita funkce aktivace je velmi důležitá, protože kdyby neurony byly lineárními prvky (pomocí lineární aktivační funkce), pak jakákoli sekvence neuronů by také produkovala lineární transformace a celá neuronová sít by pak byla ekvivalentní jednomu neuronu [8].

V posledních letech velkou popularitu získala funkce aktivace, která se nazývá ***Rectified Linear Unit*** ($ReLU$). Tato funkce je definována vzorcem:

$$f(x) = \max(0, x), \tag{1.2}$$

kde x je vstup do neuronu.

ReLU změní záporné hodnoty na nuly a kladné hodnoty beze změny převede na výstup. Aplikace této funkce výrazně zlepšuje rychlost trénování konvolučních sítí [9].

Softmax funkce je výstupní logická funkce. Používá se při klasifikačních úlohách, kde objekt (obrázek) by měl patřit jen do jedné hromadné třídy. Přiřazuje pravděpodobnosti všem výstupům sítí. Například jaká by byla pravděpodobnost, že vstupní obrázek patří pravě do skupiny aut, kol, letadel atd. Hodnoty jsou v intervalu (0,1) a součet všech výstupů by tedy měl být roven 1. Funkci softmax lze zapsat matematicky:

$$y_i = \frac{\mathrm{e}^z}{\sum\limits_{k=1}^{k} \mathrm{e}^z}. \tag{1.3}$$

Tato funkce vypočítá podíl každého výstupu tím, že ho vydělí součtem všech výstupů. Kde y_i je součet všech aktivací neuronů a je roven 1, e^z je aktivační funkce [10].

1.3 Učení s učitelem

Aby síť mohla rozpoznávat nebo klasifikovat, potřebujeme jí to naučit. Existuje hodně učících algoritmů, ale pro učení konvolučních neuronových sítí se nejčastěji používá metoda ***učení s učitelem***. Příkladem je situace, kdy se síť učí pomocí vstupního vektoru dat a učitele. Kde jako takzvaný učitel slouží vektor výstupních hodnot, ke kterému náleží vektor vstupních hodnot. V průběhu učení je hodnota vstupního vektoru převedena na vstupní vrstvu sítě. Následně je tato hodnota sítí zpracována a převedena na výstupní vrstvu, kde je porovnána s příslušnou hodnotou výstupního vektoru. Jinými slovy můžeme říct, že u metody učení s učitelem je požadovaný výstup vždy známý a síť má k dispozici vzor odpovídajícího chování, tedy zná vhodnou transformaci vstupních vektorů na výstupní vektory. Během této transformace se počítá chybová funkce, která se vyjadřuje pomocí ***střední kvadratické odchylky***:

$$f(w) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - a_n)^2, \tag{1.4}$$

kde $f(w)$ je chybová funkce, x_n je n-tý vstupní vektor s délkou N, $y(x_n, w)$ je aktuální výstup neuronové sítě závislý na odpovídajícím vstupu a hodnotách vah w a a_n je výstup, který očekáváme. V případě takové odchylky mezi výstupní a očekávanou hodnotou, se budou měnit váhy v souladu s algoritmem, snažícím se tuto odchylku minimalizovat. Váhy se budou měnit, dokud odchylka učení nedosáhne přijatelně nízké úrovně [11].

Jedním z podobných algoritmů je algoritmus klesání podle gradientu nazývaný jako ***Gradient Descent***. Gradient se počítá jako součet parciálních derivací chybové funkce. Jak již bylo naznačeno metody, jako klesání podle gradientu využívají chybovou funkci, aby se minimalizovala její hodnota a to úpravou vah sítě tak, aby se funkce postupně blížila ke svému lokálnímu minimu. Pohyb ve směru gradientu zvyšuje hodnotu funkce (zhoršuje výsledky sítě) a pohyb proti směru gradientu snižuje hodnotu funkce (zlepšuje síť). Minimalizace chyby se tedy bude počítat ve směru opačném vzhledem ke gradientu [12].

Při přiřazení nových hodnot vahám sítě se zavádí nový parametr nazývaný ***koeficient rychlosti učení*** (*learning rate*), který určuje velikost kroku při úpravě vah. Tento parametr může nabývat hodnot od 0 do 1. V případě zvolení 0 se žádné úpravy vah neprovádí. Vhodná volba tohoto parametru závisí na potřebách aktuální aplikace. Při nastavení kroku v rozsahu například 0,7 až 1 bude tento krok

dostatečně velký, což vede ke snížení počtu potřebných kroků k dosažení minima (rychlejší zpracovaní), nicméně toto zrychlení má za následek zvětšení chybovosti vzniklých vah. Zatímco, když zvolíme hodnoty v intervalu 0,1 až 0,3 počet kroků i délka trvání zpracování se několikanásobně zvětší a výsledek učení v tomto případě bude přesnější. Běžně se konvoluční neuronové sítě trénují na velikém počtu dat. Počítat pro celý trénovací dataset je moc dlouhý proces, občas i nemožný na jednom počítači. Kvůli tomu by použití metody gradient descent zpomalilo celý proces. Obvyklým řešením tohoto problému se pro zrychlení výpočtu využívá metoda ***stochastické klesání podle gradientu*** (*stochastic gradient descent*). Na rozdíl od první varianty se váhy budou upravovat jen u podmnožiny trénovacích dat. Při velkém množství analyzovaných vzorků se získané výsledky blíží výsledkům obecného gradient descent.

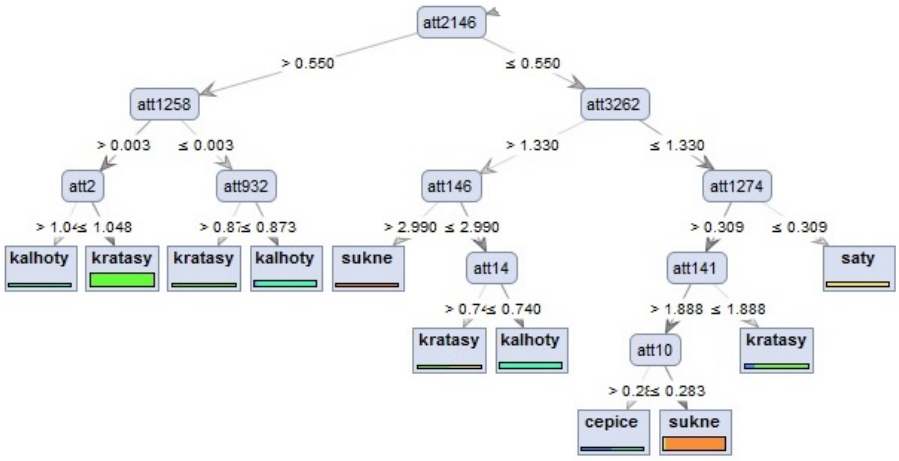
Zpětné šíření chyby (*backpropagation*) je efektivním algoritmem pro výpočet těchto gradientů. Tato metoda spočívá v opakování dvou cyklů, jako propagace a aktualizace vah. Při dopředném průchodu skrz celou síť se vypočítají gradienty chybových funkcí na výstupu a to na základě již zmíněných metod pro hledaní minima chybových funkcí. Následně tuto chybu algoritmus propaguje zpět sítí až ke vstupní vrstvě, kdy se v průběhu tohoto procesu modifikují i váhy.

2 KLASIFIKAČNÍ ALGORITMY

Všechny algoritmy popsané v této kapitole patří ke klasifikačním algoritmům strojového učení a spadají pod oblast učení s učitelem, kde jako takzvaný učitel slouží data s informacemi o tom, ke které skupině objekt patří.

2.1 Rozhodovací strom (Decision Tree)

Rozhodovací stromy se používají při roztřiďování dat do určitých skupin. Tyto stromy představují graf s hierarchickou, na sebe navazující strukturou a umožňují znázornit rozvinutí alternativních řešení určité otázky. Rozhodovací strom se podobá skutečnému, ale vzhůru přetočenému stromu, protože kořen stromu je vrcholem, a rozvíjí se směrem dolů. Postupně od kořene se vstupní data odvíjí na malé podmnožiny takzvané uzly. Každá větev je stanovená určitou podmínkou uzlu. Vstupní data se dělí do tříd podle informaci, které jsou v tomto uzlu zahrnuty. Uzel od kterého nevede žádná větev, se nazývá list, a je konečným prvkem rozhodovacího stromu. Tyto listy reprezentují třídy, do kterých probíhá klasifikace. Hloubkou stromu se nazývá počet uzlů mezi kořenem a listem. Část rozhodovacího stromu je uvedená na obr. 2.1. Pomocí tohoto procesu lze snadně interpretovat vstupní data, což je velkou výhodou této metody [15] [16].



Obr. 2.1: Rozhodovací strom

2.2 Náhodný les (Random Forest)

Hlavní myšlenkou této metody je to, že několik rozhodovacích stromů bude rozhodovat o tom, jestli určitý objekt patří do klasifikační třídy. Jinými slovy skupina

rozhodovacích stromů vytváří náhodný les. Nejpoužívanější algoritmus pro vytváření náhodných lesů je tak zvaný bagging. Tento algoritmus umožňuje vytvářeni více trénovacích podmnožin s náhodně vybranými prvky z jednoho základního trénovacího souboru. Každá z takhle formovaných podmnožin je pak využitá jedním z více rozhodovacích stromů pro natrénování. Všechny natrénované stromy se pak navzájem náhodně liší a nejsou závislé mezi sebou. Při klasifikaci, projde neznámý objekt každým rozhodovacím stromem v lese, jednotlivé stromy pak „hlasují“ do jaké z tříd daný objekt patří a následně se vybírá třída s největším počtem hlasů [17].

2.3 *K*-nejbližších sousedů (*k*-nearest-neighbor)

Tento algoritmus se řadí do jednoho z nejjednodušších algoritmů strojového učení a je založen na analogii objektu. Všechny objekty z předloženého trénovacího souboru tvoří body v n-dimenzovaném prostoru. Hodnota n je daná počtem atributů (číselných hodnot, tvořících vektory), představujících určité příznaky zkoumaných objektů. Pomocí těchto atributů lze také určit polohy daných bodů. Myšlenka tohoto algoritmu spočívá v tom, že objekty, které patří do stejné třídy, jsou rozmístěné blízko k sobě. To znamená, že v průběhu klasifikace se poloha klasifikovaného objektu porovnává s polohami objektů z trénovacího souboru. Algoritmus vrací hodnotu, jako svůj odhad, což je nejčastěji se vyskytující objekt mezi k trénovacími objekty blízkými tomu klasifikačnímu, proto je metoda nazývána jako metoda k-nejbližších sousedů [18] [19].

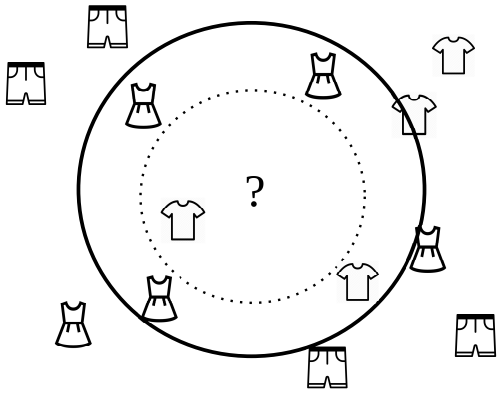
Takzvaný nejbližší soused v n-dimenzovaném prostoru je pak stanoven pomocí euklidovského vzorce:

$$\delta(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}, \tag{2.1}$$

kde δ je vzdálenost bodů (vektorů), A a B jsou prvky, A_i a B_i označují hodnotu $i - t\acute{e}ho$ atributu bodu A nebo B.

Pro lepší pochopení této metody na obr. 2.2 je zobrazen princip k-NN. Pokud je k=1 klasifikační objekt zařadíme do třídy tričko, do třídy šaty pak při zvolení k=5.

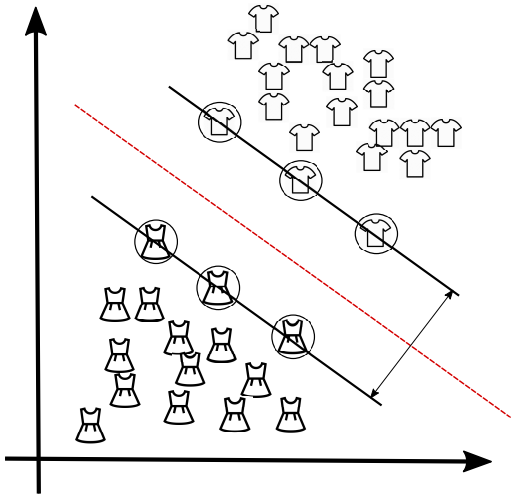
Jednou z mála výhod této metody je jednoduchost. Největší nevýhodou je čas pro klasifikaci, který lineárně vzrůstá s velikostí trénovacích dat. Také může dojít k situaci, kdy podstatné atributy jsou stejné, ale ostatní hodnoty se liší a tím pádem objekty budou velmi vzdálené a dojde k chybné klasifikaci. Lepších výsledků lze dosáhnout při úlohách s menším počtem atributů.



Obr. 2.2: Princip metody K-nejbližších sousedů

2.4 Metoda podpůrných vektorů (Support Vector Machnine)

Metoda podpůrných vektorů patří k relativně novým metodám strojového učení. Jako u metody k-nn se objekty z trénovacího souboru ukládají do n-dimenzovaného prostoru. Ale princip SVM je založen na transformaci výchozích vektorů do prostoru s větší dimensionalitou, ve které se dají lépe rozdělit na třídy. Prakticky se po transformaci hledá nadrovina, která bude optimálně rozdělovat prvky trénovacích dat na základě lineární funkce. Z hlediska přesnosti klasifikace se vybere taková nadrovina, při které je vzdálenost mezi rozdělenými třídami maximální. Vektory, které se nacházejí nejblíž k rozdělující nadrovině se nazývají podpůrné a slouží jako oddělovací podpora nadroviny [20].



Obr. 2.3: Metoda podpůrných vektorů

2.5 Naivní bayesovský klasifikátor (Naive Bayes Classifier)

Tento algoritmus představuje jednoduchý pravděpodobnostní klasifikátor, který je založen na aplikaci bayesovského teorému o pravděpodobnosti viz vzorec 2.2.

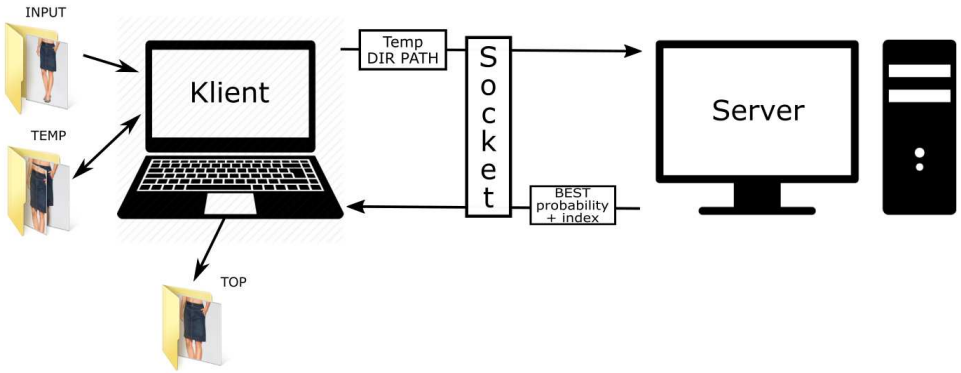
$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)}, \tag{2.2}$$

kde $P(X)$, $P(Y)$ je pravděpodobnost výskytu X nebo Y a $P(X \mid Y)$, $P(Y \mid X)$ je pravděpodobnost výskytu X nebo Y za předpokladu výskytu Y nebo X.

NBC vychází z předpokladu, že jednotlivé příznaky objektu klasifikace nejsou závislé mezi sebou. Jinak řečeno tento klasifikátor předpokládá, že vyskytnutí (nebo absence) určitého atributu dané třídy nezáleží na existenci (absenci) nějakého dalšího příznaku té samé třídy. Kvůli tomuto zjednodušení je tento klasifikátor nazýván „naivní“. Například oblečení považujeme za bundu, pokud má dva rukávy, zapínaní a třeba kapuci. I kdyby tyto příznaky byly závislé na nějakých dalších, Naivní bayesovský klasifikátor předpokládá, že všechny tyto atributy samostatně napomáhají pravděpodobnosti toho, že zkoumané oblečení je bunda [21] [22].

3 NÁVRH A IMPLEMENTACE

Program implementovaný v rámci této bakalářské práce, jak již bylo zmíněno, byl z důvodu snížení hardwarových požadavků rozdělen na část klienta a serveru. Na obr. 3.1 je zakresleno schéma tohoto programu. Postup analýzy obrázku bude rozebrán v následujících kapitolách.



Obr. 3.1: Program klient-server

3.1 Instalace a tvorba instalátoru

Pro práci s konvolučními neuronovými sítěmi používanými v rámci této bakalářské práce byl zvolený software Anaconda. Před instalací samotného programu Anaconda byl nejdříve nainstalován open-source kompiler pro Windows TDM-GCC. Program Anaconda slouží pro práci s daty a na vědecké výpočty, používá se také pro strojové a hluboké učení. Tento nástroj je bezplatný, snadně se instaluje, je distribucí Pythonu, zahrnuje sbírku více než 150 balíčků, které jsou nainstalované automaticky, a také více než 250 balíčků, které můžou být nainstalované jednoduchým příkazem `conda install`. Zahrnutí balíčkovacího systému `pip` nám pak umožnilo nainstalovat pomocí příkazů:

```
pip install git+git://github.com/Theano/Theano.git
pip install git+git://github.com/fchollet/keras.git
```

potřebné knihovny Theano a Keras. Pro použití těchto příkazů byla také nutná instalace aplikace GitHub, protože příkazy vyžadují funkcionalitu úložiště kódu github.

- ***Theano***

  Kvůli složitosti výpočetních algoritmů, neuronové sítě vyžadují optimální matematické algoritmy, které bývají ukládané do knihoven. Podobnou specializovanou knihovnou je knihovna Theano. Theano je rozšíření jazyka Python, které umožňuje efektivní výpočty matematických výrazů obsahujících i vícerozměrné matice. Také těsně spolupracuje s knihovnou NumPy pro vědecké výpočty, obsahující algoritmy lineární algebry, Fourierovy transformace atd. Theano poskytuje možnosti výpočtu jak na CPU, tak i na GPU (pomocí NVIDIA CUDA) a to beze změny programového kódu. Je to otázkou přepisu položky `device` na „gpu“ nebo „cpu“ v souboru `theanorc`, který se vytvoří sám po přidání knihovny. V případě použití GPU je zapotřebí ještě přidat cuDNN (výkonná knihovna pro strojové učení s funkcionalitou užívanou sítěmi hlubokého učení, poskytuje optimalizovanou verzi funkcí jako například konvoluce). Theano pracuje na Windows, ale ještě vyžaduje instalaci doplňkových nástrojů. Tyto nástroje jsou:
  - kompiler pro C/C++ (pro Python verze 2.7 je to kompiler Microsoft Visual Studio do verze 2013);
  - CUDA (verzi, která podporuje nainstalované Visual Studio);
  - GCC (pro kód, který nepoužívá CUDA).
- ***Keras***

  Keras je minimalistickou knihovnou Pythonu pro hluboké učení, která běží na Theanu. Byla vyvinuta se zaměřením na možnost rychlého experimentování s neuronovými sítěmi, protože rychlost, za kterou se dobereme výsledku, je zde klíčová. Jádrem datové struktury Kerasu je model, který je způsobem organizace vrstev. Hlavním typem modelu je *Sequential* model, což je model s lineární skupinou vrstev.

V rámci této bakalářské práce byl vytvořen automatizovaný instalátor, který obstarává instalaci všech výše zmíněných aplikaci. Tento instalátor byl vyvinut pomocí instalačního průvodce pro prostředí operačních systémů Microsoft Windows, který se nazývá *INNO Setup*. Tento program se vyznamenává poměrně velkou snadností pochopení vytváření instalačních průvodců. Princip jejich fungování se definuje pomocí skriptu, na základě kterého se pak vygeneruje spustitelný soubor s instalačním průvodcem. Instalátor samotný umístí na vybrané místo všechny potřebné instalační soubory včetně jednoho dávkového souboru, od nějž umístí na ploše uživatele zástupce. Tento dávkový soubor po spouštění nainstaluje veškeré potřebné programy pro správnou funkci umělých neuronových sítí.

3.2 Extrakce dat z předposlední vrstvy

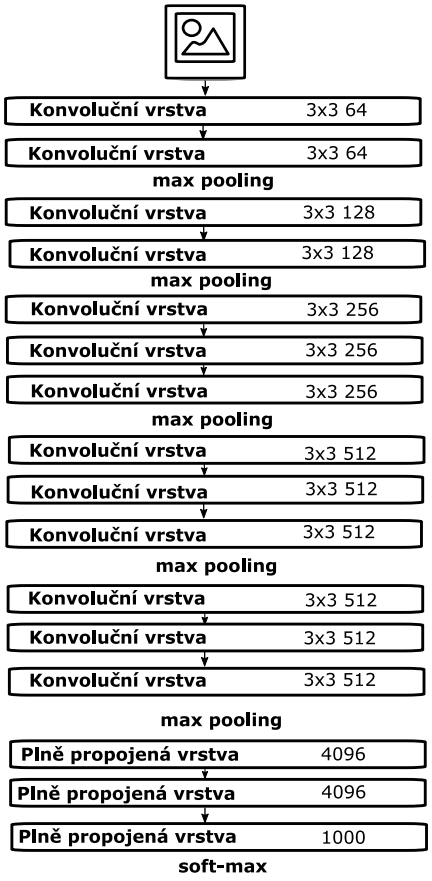
Hlavním úkolem této bakalářské práce bylo z počátku nalézt možnosti vyčítání hodnot z vrstev umělých neuronových sítí, zejména z předposlední vrstvy modelu. Poslední vrstva je schopná přiřadit dodaný obrázek do určité skupiny, zatímco předposlední vrstva obsahuje požadované informace o nízkoúrovňových příznacích z obrazu. Pro extrakci příznaků byl navrhnut program na bázi klient-server.

Část serveru byla navržená v jazyce Python. V počátcích vývoje byl pro úkol vyčítání hodnot z obrázku použit již předtrénovaný model hluboké neuronové sítě s názvem VGG16. Tento model byl navržen Karen Simonyan a Andrew Zisserman v [13]. Při trénování daného modelu se jedná o učení s učitelem, musí tedy existovat trénovací data. V tomto případě je síť natrénovaná na sadě obrázků z webu ImageNet. ImageNet je databází obrázků, obsahuje více než 14 milionů obrázků seřazených do 1000 tříd. Většina názvů těchto tříd zatím odpovídá podstatným jménům.

V použitém modelu procházel vstupní obrázek skupinou třinácti konvolučních vrstev s konvolučními filtry o velikosti 3×3. Krok posunu filtru je stanovený na 1 pixel. Část z konvolučních vrstev pak následují vrstvy max-poolingu, které používají filtry 2×2 s krokem rovným 2 pixelům. Dále po soupravě těchto vrstev jsou tři plně propojené vrstvy. První dvě plně propojené vrstvy mají 4096 kanálů, třetí plně propojená vrstva obsahuje 1000 neuronů odpovídajících 1000 klasifikačním třídám a je klasifikována pomocí aktivační funkce *softmax*. Na každé skryté vrstvě je aplikovaná nelineární aktivační funkce ReLU. Pro správné fungování sítě bylo také nutné stáhnout soubor s nastavenými váhami. Diagram použitého modelu je zobrazen na obr. 3.2.

Dalším nezbytným krokem bylo vhodné navržení způsobu uskutečnění přenosu dat mezi klientem a serverem. Pro tento účel byl vytvořen socket. Data se budou posílat pomocí TCP na port 5008 s adresou localhost pravě používaného počítače 127.0.0.1. Port 5008 využíváme pro tvorbu tunelu mezi klientem a serverem. Pomocí metody `s.listen(1)` spouštíme pro daný socket režim naslouchání, 1 znamená, že metoda přijímá pouze jeden argument tj. maximální počet připojení. Užitím `s.accept` se navazuje spojení, tato funkce vrací pár `(conn, address)`, kde `conn` je nový socket a `address` je adresou klienta. Pravě tento nový socket bude použit pro příjem a odeslání dat klientovi. V tomto okamžiku je navázané spojení mezi klientem a serverem. Kvůli tomu, že server nemůže přesně vědět, co mu klient pošle a jak velké to bude, budeme od něj přijímat data v malých dávkách. Chceme-li získat data, je třeba použít metodu `s.recv`, která má jako argument maximální počet bajtů ke čtení. Budeme číst dávky po 1024 bajtech. Po dokončení aktuální úlohy se spojení ukončí metodou `conn.close`.

Z důvodu urychlení práce byl pro spouštění serveru samotného napsán jedno-



Obr. 3.2: Diagram použitého modelu konvoluční neuronové sítě

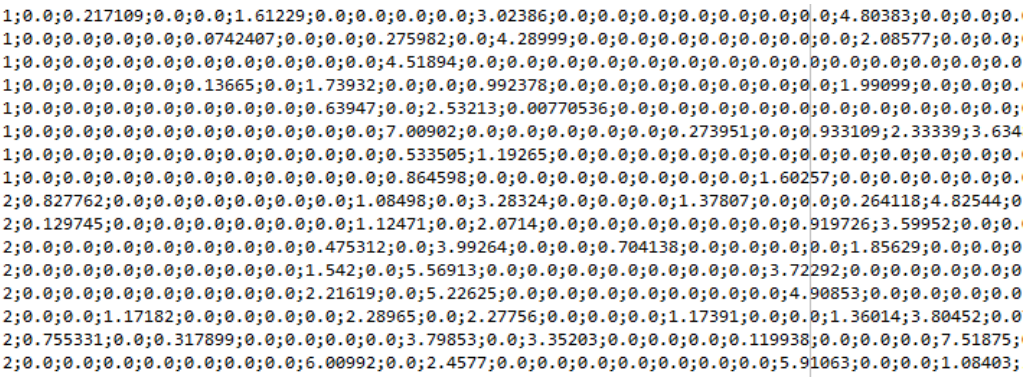
duchý dávkový soubor pro Windows, který inicializuje server v konzolovém okně. V konzolovém okně se v průběhu analýzy vypisují informace o aktuálně zpracovávaném obrázku a o stavu serveru viz obr. 3.3. Server po spuštění čeká na připojení klienta, poté s použitím vytvořené neuronové sítě zpracovává poslaná data a nakonec se do klientské části odesílá odpověď ve formě 4096 různých hodnot oddělených středníkem pro další zpracování. Hodnoty odpovídají všem výstupům neuronu na předposlední plně propojené vrstvě.

Klientská část je napsána v jazyce Java a spouští se přímo přes vývojové prostředí Eclipse. Pomocí vytvořeného cyklu posílal první klient postupně na server obrázky z vybraných složek. Bylo vytvořeno celkem 10 složek s nejčastěji se vyskytujícími prvky oblečení jako jsou tričko, kalhoty, šaty atd. V každé složce je 100 obrázků odpovídajících těmto prvkům.

Program procházel každou složku a postupně posílal na server ke zpracování všechny obrázky z každé složky. Výsledky dodané serverem byly uloženy do textového souboru, ve kterém každý řádek odpovídal jednomu obrázku ze složky. Také každé skupině oblečení (kalhoty, šaty, atd.) odpovídalo číslo na začátku řádku. Na

Obr. 3.3: Ukázka výpisu informací v konzoli

obr. 3.4 je vidět, v jakém formátu byla data obdržená od serveru uložená klientem. Pak tyto hodnoty posloužily k analýze jednotlivých klasifikačních algoritmů pro popis možných vyskytujících se částí oblečení na předložených obrazcích.



Obr. 3.4: Ukázka textového souboru

Prvním nalezeným způsobem pro samotnou extrakci dat z předposlední vrstvy je příkaz `model.pop()`. Pomocí tohoto příkazu se před kompilaci modelu odstraní poslední vrstva. Dále použitím Keras příkazu `model.predict(im)` extrahujeme data z vytvořeného modelu bez poslední vrstvy. Příkaz `model.pop()` nám umožňuje smazaní vrstev od konce modelu. Ale v případě, že bychom chtěli extrahovat data opakovaně z různých jiných vrstev jednoho modelu, dala by se použít Keras-funkce, která dokáže vyčíst hodnoty z vrstev nedestruktivním způsobem tj. bez nenávratného poškození modelu. Pokud tedy chceme analyzovat model bez poslední vrstvy bude K-funkce vypadat následovně

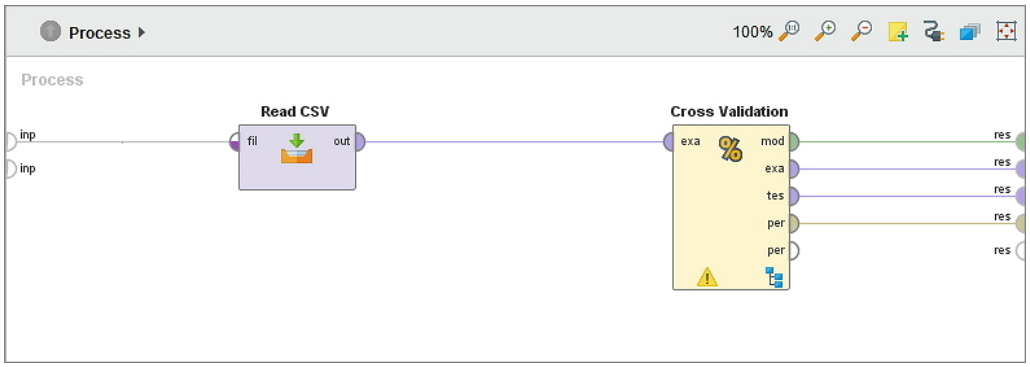
```
K.function([model.layers[0].input, K.learning_phase()],
[model.layers[34].output])
```

Jelikož tato funkce neovlivňuje model jako takový, můžeme tedy následně provést analýzu i jiného počtu vrstev. Toho bychom následně dosáhli pouze změnou čísla počáteční a konečné vrstvy ve zmíněném příkazu.

3.3 Analýza extrahovaných dat

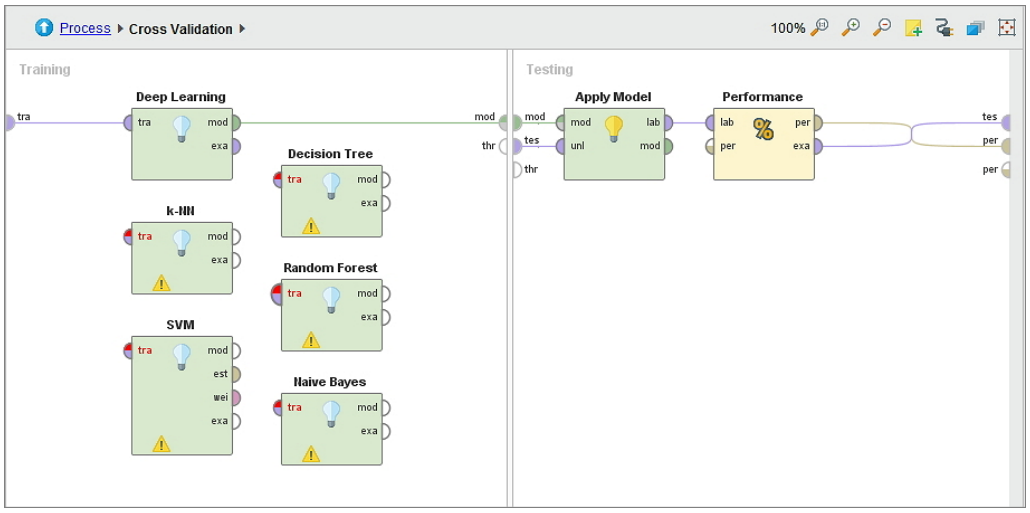
Pro analýzu extrahovaných dat byl použit nástroj Rapidminer. V současné době je Rapidminer jedním z nejpoužívanějších a nejrozšířenějších open-sorce programů pro analýzu dat. Tento software poskytuje možnosti pro dolování dat a také strojové učení. Celý program je napsaný v programovacím jazyce Java. Nabízí grafické uživatelské rozhraní, které umožňuje navrhnout a uskutečnit analytické pracovní postupy. Tyto postupy jsou nazvány *Procesy* v Rapidmineru a skládají se z několika *Operátorů*. Každý operátor provádí jednotlivé úkoly v rámci jednoho procesu a má na levé straně vstup a na pravé výstup. Výstup každého operátoru je propojen s vstupem operátoru následujícího. Pomocí čehož je vytvořen celý proces. [14]

V této práci bylo vytvořeno zapojení znázorněné na obr. 3.5



Obr. 3.5: Ukázka schéma zapojení

Do prvního zapojeného operátoru Read CSV byla nahrána extrahovaná data z obrázků ve formátu CSV (Comma-separated values). Tento formát je vhodný pro výměnu tabulkových dat sestávajících z řádků, ve kterých hodnoty, jak jde pochopit z názvu, jsou oddělené tečkou. Jelikož se v češtině čárka používá jako oddělovač desetinných míst, je možné použít také středník jako v našem případě. Také pomocí tohoto operátoru byl první sloupec s názvem oblečení označen jako label pro námi žádoucí výstup. Tento soubor obsahuje 1000 řádku (10 druhů oblečení×100 obrázků pro každý druh). Na výstup předložených dat byl aplikován operátor Cross Validation (křížová validace) viz obr. 3.6. Při křížové validaci jsou vstupní data rozdělená na k na sebe nezávislých podmnožin stejné velikosti (obvykle 10). Tento operátor také zahrnuje dva podprocesy: proces trénování a testovací proces. Tyto



Obr. 3.6: Obsah operátoru Cross Validation

	A *polynominal* *label*	B *real*	C *real*	D *real*	E *real*	F *real*
1	bunda	0.000	0.965	0.000	0.000	1.949
2	bunda	0.000	0.000	0.118	0.000	0.000

Obr. 3.7: Ukázka labelu

procesy proběhnou *k*-krát, přičemž *k-1*podmnožin poslouží pro trénovaní a zbylá 1 podmnožina pro testovaní. Tento proces se bude opakovat dokud každá z podmnožin nebude použitá pro testování a v ostatních cyklech pro trénování. Účelem křížové validace je vytvoření modelu, který bude mít největší přesnost. Trénovací část tohoto operátoru obsahuje jednotlivé metody strojového učení, které jsou popsané v kap.2. Poté je model otestován. Operátor Performance slouží ke stanovení úspěšnosti výsledků natrénovaného modelu. Jednou z možností posouzení přesnosti natrénovaného modelu je matice záměn (confusion matrix) a je součástí výstupu operátoru Performance. Matice záměn se používá pouze u klasifikačních úloh. Velikost této matice je n×n a je daná počtem tříd, pro které se provádí klasifikace (v tomto případě matice má velikost 10×10). Sloupce představují skutečné třídy, zatímco řádky jsou třídy, které byly klasifikované použitým algoritmem strojového učení. Hodnoty na různých pozicích matice potom ukazují, kolik zástupců jednotlivých skupin podle klasifikátoru patří právě těmto skupinám. Je zřejmé, že nejlepší výsledky budou dosažené v případě, když většina těchto hodnot bude ležet na hlavní diagonále a hodnoty mimo diagonálu budou nulové.

3.3.1 Klasifikace pomocí vybraných metod

V této podkapitole jsou ukázané výsledky jednotlivých klasifikačních algoritmů pomocí matic záměn. Také zde jsou uvedeny položky nastavení, které byly ponechány ve výchozích nastaveních.

Hluboké učení

Activation (funkce aktivace): ***ReLU***
Hidden layer sizes (počet skrytých vrstev): ***16***
Hidden dropout ratios (hodnota Dropoutu): ***0,5***
Epochs (počet epoch): ***20***
Train samples per iteration (počet vzorků pro jednu iteraci): ***20***
Loss function (typ ztrátové funkce): ***CrossEntropy***

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	86	3	6	2	13	0	0	1	1	2	75,7%
čepice	0	94	0	1	0	0	01	0	0	0	98,9%
kabát	4	0	86	0	3	2	0	6	1	0	84,6%
kalhoty	0	1	0	94	0	2	0	1	2	0	93,8%
košile	9	2	4	1	85	1	0	0	0	9	75,5%
kraťasy	0	0	0	1	0	94	0	2	3	0	93,9%
plavky	0	0	0	0	0	0	97	1	0	0	99,0%
šaty	0	0	1	1	0	0	1	87	2	3	91,5%
sukně	0	0	1	2	0	2	0	2	93	0	92,9%
tričko	0	1	0	1	4	0	0	1	0	91	92,5%
třída	86%	94%	86%	94%	85%	94%	97%	87%	93%	91%	

Celková přesnost 90,7%

Tab. 3.1: Výsledek algoritmu hlubokého učení

Rozhodovací strom

Maximal depth (maximální hloubka stromu): ***20***
Confidence (jistota předpovědi): ***0,25***
Minimal gain (minimální hodnota pro dělení uzlu): ***0,1***

Minimal leaf size (minimální počet listů každé třídy): ***2***

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	67	3	20	1	10	1	0	1	0	1	64,4%
čepice	3	71	3	2	0	0	1	7	1	0	80,7%
kabát	13	5	59	2	8	0	0	3	1	1	64,1%
kalhoty	1	3	2	71	4	13	1	4	9	2	64,6%
košile	14	4	5	3	68	3	0	2	0	9	62,9%
kraťasy	0	3	0	7	1	76	0	1	8	1	78,4%
plavky	1	2	0	1	0	1	90	1	1	0	92,8%
šaty	0	6	6	3	1	1	5	70	10	6	64,8%
sukně	0	1	3	8	0	4	2	11	68	1	69,4%
tričko	1	2	2	2	8	1	1	0	2	79	80,6%
třída	67%	71%	59%	71%	68%	76%	90%	70%	68%	79%	

Celková přesnost 71,9%

Tab. 3.2: Výsledky algoritmu Rozhodovací strom

Naivní bayesovský klasifikátor

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	47	0	8	3	21	1	0	0	0	4	55,9%
čepice	13	100	23	10	13	8	9	11	8	7	49,5%
kabát	19	0	68	5	5	2	0	3	3	1	64,2%
kalhoty	0	0	0	60	0	6	0	0	1	0	89,6%
košile	20	0	0	0	52	0	0	0	0	3	69,3%
kraťasy	0	0	0	15	0	77	0	0	2	0	81,9%
plavky	0	0	0	0	0	0	89	01	0	0	100,0%
šaty	0	0	1	3	0	5	2	84	31	8	62,8%
sukně	0	0	0	4	0	0	0	0	54	0	93,1%
tričko	1	0	0	0	9	1	0	2	1	77	84,6%
třída	67%	71%	59%	71%	68%	76%	90%	70%	68%	79%	

Celková přesnost 70,8%

Tab. 3.3: Výsledek metody NBC

K-nejbližších sousedů

k (počet vybraných trénovacích objektů nejbližších k neznámému): ***1***
Measure types (typ měření): ***Mixed measure***
Mixed measure (typ smíšeného měření): ***Mixed Euclidean Distance***

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	73	1	15	0	6	0	0	0	0	1	75,4%
čepice	0	98	0	0	0	0	0	1	0	0	98,9%
kabát	4	0	74	0	0	0	0	4	0	0	90,2%
kalhoty	0	0	1	91	1	3	0	1	5	0	89,2%
košile	21	0	7	0	86	1	0	0	0	11	68,3%
kraťasy	0	0	0	8	0	94	0	1	5	0	87,0%
plavky	0	0	0	0	0	0	100	2	0	0	98,0%
šaty	0	0	3	0	0	0	0	88	1	0	95,7%
sukně	0	1	0	1	0	2	0	2	89	0	93,7%
tričko	2	0	0	0	7	0	0	1	0	88	89,8%
třída	73%	798%	74%	91%	86%	94%	100%	88%	89%	88%	

Celková přesnost 88,1%

Tab. 3.4: Výsledek metody k-NN

Náhodný les

Number of trees (počet stromů): ***10***
Maximal depth (maximální hloubka stromu): ***20***
Confidence (jistota předpovědi): ***0,25***
Minimal gain (minimální hodnota pro dělení uzlů): ***0,1***
Minimal leaf size (minimální počet listů každé třídy): ***2***

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	43	15	34	1	28	5	0	7	4	5	32,7%
čepice	3	49	3	2	2	4	7	10	3	5	55,7%
kabát	16	2	31	4	10	1	0	3	3	5	41,3%
kalhoty	7	9	9	65	14	19	1	5	15	12	41,7%
košile	14	5	9	0	26	2	0	2	1	10	37,7%
kraťasy	0	0	0	16	3	36	2	0	10	6	49,3%
plavky	1	13	1	6	2	11	83	21	9	10	52,9%
šaty	1	2	5	0	0	2	5	38	13	7	52,1%
sukně	4	1	6	3	4	9	2	11	33	1	44,6%
tričko	6	4	2	3	11	11	0	3	9	39	80,6%
třída	48%	49%	31%	65%	25%	36%	83%	38%	33%	39%	

Celková přesnost 44,8%

Tab. 3.5: Výsledky algoritmu Náhodný les

Metoda podpůrných vektorů

Convergence epsilon (parametr pro optimalizaci): ***0,001***

Max iterations (počet opakovaní): ***1000***

	Skutečný výsledek										
odhad	bun.	čep.	kab.	kal.	koš.	kra.	pla.	šat.	suk.	tri.	přesnost
bunda	87	0	6	0	5	0	0	0	0	0	88,8%
čepice	0	99	0	0	0	0	0	1	0	0	99,0%
kabát	2	1	86	0	0	0	0	1	0	0	95,6%
kalhoty	0	0	1	97	1	3	0	3	2	0	90,7%
košile	1	0	6	0	91	1	0	1	0	7	77,8%
kraťasy	0	0	0	1	0	95	0	0	3	1	95,0%
plavky	0	0	0	0	0	0	100	3	0	0	97,1%
šaty	0	0	1	0	0	0	0	83	0	0	98,8%
sukně	0	0	0	2	0	1	0	6	95	0	91,4%
tričko	0	0	0	0	3	0	0	2	0	92	94,7%
třída	87%	99%	86%	97%	91%	95%	100%	83%	95%	92%	

Celková přesnost 89,5%

Tab. 3.6: Výsledek SVM algoritmu

Porovnání klasifikačních algoritmů

V tabulce 3.7 lze vidět shrnutí výsledků získaných z jednotlivých klasifikačních algoritmů. Dle dosažených výsledků se dá usoudit, že metoda hlubokého učení je pro tento zadaný úkol klasifikace nejvhodnější a to kvůli úspěšnosti téměř 91%. Metoda podpůrných vektorů a metoda K-nejbližších sousedů zde také dosáhly příznivých výsledků a to okolo 90%, zatímco nejhorší výsledky obdržela metoda Náhodný les a to pouze 44,8%.

	Metoda	Úspěšnost
1	Rozhodovací strom	71,9%
2	Náhodný les	44,8%
3	K-nejbližších sousedů	88,1%
4	Metoda podpůrných vektorů	89,5%
5	Naivní bayesovský klasifikátor	70,8%
6	Hluboké učení	90,7%

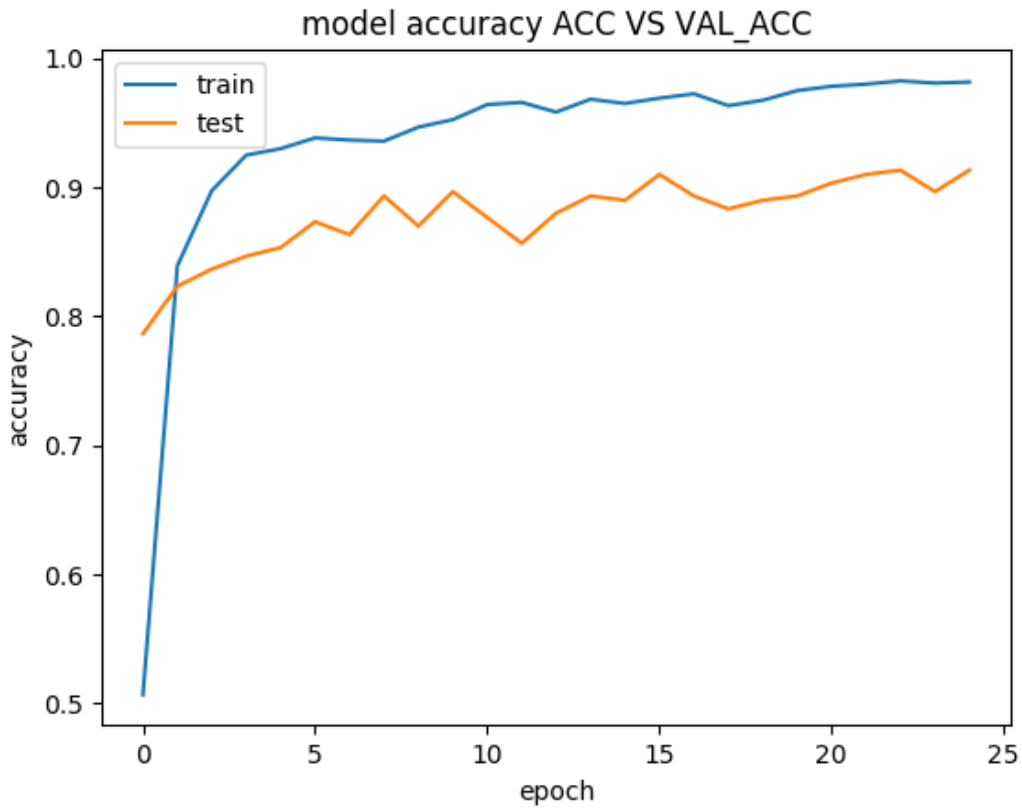
Tab. 3.7: Výsledky jednotlivých metod

3.4 Natrénovaní vlastního modelu neuronové sítě

Při natrénování modelu neuronové sítě byl využíván notebook s `CPU Intel(R) Core(TM)i-5-2430M 2,4GHz, RAM 8GB a GPU NVIDIA GEFORCE GT540M 2GB`.

Prvním nezbytným krokem při trénování je příprava trénovacích a testovacích dat. Tato data jsou umístěná v jedné jediné složce nazvané `input data`. Data z této složky jsou následně zpracovaná, převedená na požadovaný rozměr a do černobílého formátu. Důvodem pro převod do černobílého formátu je to, že barevné kusy oblečení by mohly být špatně klasifikované kvůli nedostatku trénovacích vzorků stejné barvy. Upravené obrázky jsou poté uloženy do pole. Obsahu tohoto pole jsou potom přiřazeny labely, které definují klasifikační třídy jako takové. Pro každý label připadá 80% vzorků na proces trénování, zbylých 20% pak na testování. V našem případě bylo z důvodu nároků na paměť počítače použito pouze 100 obrázků pro každý label. Cyklus trénování je rozdělen na tzv. epochy. Epochy jsou intervaly, v nichž proběhne předložení všech trénovacích vzorků. Tento krok je zapotřebí opakovat vícekrát pro dosažení vyšších přesností sítě. Při námi provedeném trénování bylo využito 25 epoch. Na obr. 3.8 lze vidět, že s počtem provedených epoch stoupala přesnost sítě.

V průběhu trénování byl navržen model konvoluční sítě, který je podobný modelu VGG16 a ten je popsaný v kap. 3.2. První vrstva udává rozměr 224×224, následně pak model pokračuje čtyřmi konvolučními vrstvami a dvěma vrstvami maxpoolingu. Aktivační funkce byla použita stejná (ReLU). Definování navrženého modelu je ukončeno jeho kompilací. Potom proběhne samotné trénování sítě, na jehož konci jsou uloženy vzniklé váhy.



Obr. 3.8: Závislost přesnosti na počtu epoch

Při testování modelu se ukázalo, že nejkomplikovanější pro síť bylo rozeznávat například bundu od košile (v 6 případech z 24) nebo také kalhoty od kraťasů (ve 4 případech z 15). Tyto výsledky jsou docela logické, ale také se projevilo, že sít ve 3 případech z 15 rozpoznala šaty jako kalhoty, toto mohlo být způsobeno jedním z atributů, který je společný pro obě dvě tyto třídy.

3.5 Výsledný program

Výsledný program pro rozpoznání a lokalizaci oblečení byl ponechaný v podobě klient-server. Spojení těchto části probíhá a funguje také pomocí socketu, jak bylo

popsáno v kap. 3.2. Klient načítá určitý obrázek zobrazený na obr. 3.9. Po načtení je tento obrázek rozdělen na sedm menších dílů, které jsou pak umístěné do speciální složky `TEMP` k tomu určené viz obr 3.10. Pět ze sedmi dílů obrázku má identický rozměr, činící 70% šířky a výšky původního obrázku, zbylé dva obrázky jsou pouze levá a pravá polovina originálního obrázku. Důvodem pro toto dělení na n různých dílů je ten, že objekt, který chceme klasifikovat, se nemusí nacházet na obrázku sám nebo také uprostřed.

Obr. 3.9: Původní obrázek

Obr. 3.10: Obrázek po úpravě

Cesta k této složce je odeslána na server, kde proběhne klasifikace každého z sedmi obrázků. Po zpracování všech částí serverem se vybere jedna z těchto částí s pravděpodobností. Informace o obrázku s nejvyšší pravděpodobností (název, pravděpodobnost a index) vrací server zpět klientovi k další analýze, kde jsou tato data zpracována potřebným způsobem. V případě, že tato pravděpodobnost není vyšší, než námi definovaná jako požadovaná, je tento obrázek opět rozdělen na sedm menších části a opět odeslán k poslednímu kolu zpracování. Po splnění podmínky s velikostí pravděpodobnosti nebo po druhém kole zpracování je obrázek s nejlepšími výsledky přesunut do složky `TOP`. Dále by se v tomto programu dala doplnit možnost vrácení většího počtu výsledků a tím bychom mohli docílit případného rozpoznání většího množství obdobných objektů na jednom obrázku. Získané výsledky jsou také vypsané do vývojového rozhraní Eclipse.

4 ZÁVĚR

Tato bakalářská práce byla věnovaná hlubokému učení, využívající umělé neuronové sítě. V kapitole 1.Konvoluční neuronová síť byly popsány základní principy fungování konvolučních neuronových sítí a jejich architektura, protože právě tento druh umělých neuronových sítí je využíván pro získání požadovaných dat z obrázků a také pro realizaci hlavního úkolu praktické části. V kapitole 2.Klasifikační algoritmy byly rozebrané vybrané metody strojového učení pro problematiku klasifikací.

V průběhu práce byla vytvořena databáze oblečení obsahující 1000 obrázků. Pomocí hluboké konvoluční neuronové sítě byla provedena extrakce dat (příznaků). Extrahovaná data byla uložena do textového souboru a následně pak byla využita pro ověření přesnosti klasifikačních algoritmů strojového učení za pomocí programu Rapidminer. Jako nejúspěšnější metoda se ukázala umělá hluboká neuronová síť, která dosáhla přesnosti téměř 91%. Vysoké výsledky měla metoda podpůrných vektorů a metoda k-nejbližších sousedů a to okolo 90%. Pouhých 44,8% obdržela metoda Náhodný les, což bylo nejhorším výsledkem této analýzy.

Výsledkem práce je program klient-server, který je schopný klasifikovat a lokalizovat určité objekty z dodaných obrázků. Na tyto části byl program rozdělen z důvodu zpracování velkého množství obrázků a také požadavkům na rychlost. Bez tohoto rozdělení by program musel při každém spuštění načítat celou použitou neuronovou síť od začátku. V klientské části tohoto programu se provádí načtení, rozdělení obrázků na části a odesílaní cesty k těmto částem na server. Kdežto na straně serveru program zahrnuje hlubokou neuronovou síť, pomocí které se provádí klasifikace dodaných částí obrazu a vybírá se ten s nejlepším výsledkem. Údaje o tomto jediném obrázku se pošlou zpět klientovi. Proces se celý zopakuje v případě, že pravděpodobnost výskytu nějakého objektu na obrázku nedosáhne potřebné úrovně.

První použitý model neuronové sítě vychází z modelu, který je dostupný a velmi rozšířený na internetu. Nicméně jeho použití v této aplikaci by nesplnilo požadavky, kterých chceme dosáhnout, a to rozeznávání specifických kusů oblečení. Původní model byl schopen rozeznat 1000 různých tříd, tyto třídy definovaly konkrétní objekty jako například auto, dům, kůň. Síť by tedy nebyla schopná rozpoznat konkrétní typy oblečení, ale pouze by přítomnost těchto objektů vyhodnotila obecně jako „oblečení“. Z tohoto důvodu byl natrénován nový model, který již tyto detaily rozpozná. Vzhledem k původnímu modelu byl nový zmenšen o několik vrstev, tak aby trénování tohoto modelu nevyžadovalo nadměrné množství času a strojových prostředků.

Použité neuronové sítě v programu Python využívají pro svoji správnou funkci knihovny Keras a Theano. Knihovna Theano se stará o složité matematické výpočty, jako například konvoluce. Zatímco knihovna Keras obstarává vytváření samotného

modelu, který pro tuto aplikaci potřebujeme.

V rámci této bakalářské práce bylo tedy dosaženo těchto stěžejních cílů. Vytvořená aplikace klient-server obsahující umělou neuronovou síť je schopna rozeznávat kusy oblečení. Velmi dobrých výsledků dosahuje tato síť v oblasti kusů oblečení, na které byla navržena. Tyto výsledky v některých případech dosahují více než 90%. Postup pro natrénování modelu by se také dal využít i pro jakoukoliv jinou oblast zájmu. Rozdělením na části klienta a serveru jsme docílili akceleraci zpracování o téměř 95%. Kvůli velmi komplikovanému procesu instalace všech potřebných aplikaci, knihoven a nástrojů byl vytvořen program, který zařídí kompletní instalaci těchto potřebných části pro platformu Windows.

LITERATURA

[1] VOLNÁ, E. *Neuronové sítě 1* [online]. Druhé vydání. Ostrava: Ostravská univerzita v Ostravě, 2008 [cit. 2016-11-23]. Dostupné z URL: <http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf>.

[2] VORONTSOV, K. *Umělé neuronové sítě: přednáška* [online]. [cit. 2016-11-23]. Dostupné z URL: <http://docplayer.ru/424583-Iskusstvennye-neyronnye-seti.html>.

[3] HAYKIN, S. *Neural Networks: A Comprehensive Foundation. Second. Upper-Daddle River, New Jersey 07458: Williams Publishing House, 2006. ISBN 0-13-273350-1.*

[4] LeCun, Y. a kol. *Gradient-based learning applied to document recognition. IEEE, 1998, 46.*

[5] KARPATHY, A. *Convolutional Neural Networks for Visual Recognition. : Convolutional Layer* [online]. 2016 [cit. 2016-11-23]. Dostupné z URL: <http://cs231n.github.io/convolutional-networks/#conv>.

[6] ZEILER, M. *Rob FERGUS. ZEILER, M. Visualizing and Understanding Convolutional Networks* [online]. New York University, USA: Dept. of Computer Science [cit. 2016-11-29]. Dostupné z URL: <http://www.cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf>.

[7] PAEVSKY, A. *Jak neuronová síť rozpoznává obraz. In: Neurozprávy* [online]. 2016 [cit. 2016-12-04]. Dostupné z URL: <http://neuronovosti.ru/convolutional/>.

[8] PANFILOV, P. *Úvod do neuronových sítí* [online]. [cit. 2016-11-30]. Dostupné z URL: <http://download.virtuosclub.ru/lib/nyeyroni.pdf>.

[9] SOKOLOV, E. *Neuronové sítě* [online]. 2015 [cit. 2016-10-15]. Dostupné z URL: <http://www.machinelearning.ru/wiki/images/1/1e/Sem07_ann.pdf>.

[10] BORISOV, E. *Metody učení vícevrstvových neuronových sítí* [online]. 2016 [cit. 2016-11-23]. Dostupné z URL: <http://mechanoid.kiev.ua/neural-net-backprop.html>.

[11] PETROV, P. *Základy umělých neuronových sítí* [online]. [cit. 2016-11-28]. Dostupné z URL: <http://www.ai2002.narod.ru/Bases.htm>.

[12] HRADIŠ, M. *Konvoluční neuronové sítě: přednáška* [online]. [cit. 2016-12-4]. Dostupné z URL: <https://www.superlectures.com/openalt2015/konvolucni-neuronove-site>.

[13] SIMONYAN, K. *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE RECOGNITION* [online]. Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015. [cit. 2016-12-5]. Dostupné z URL: <https://arxiv.org/pdf/1409.1556v6.pdf>.

[14] *RapidMiner*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2006 [cit. 2017-04-06]. Dostupné z URL: <https://en.wikipedia.org/wiki/RapidMiner>.

[15] *Decision Tree*. RapidMiner Documentation [online]. [cit. 2017-04-06]. Dostupné z URL: <http://docs.rapidminer.com/studio/operators/modeling/predictive/trees/parallel_decision_tree.html>.

[16] ŠACHIDI, A. *Náhodné stromy - obecné principy fungování. BASEGROUP LABS* [online]. [cit. 2016-04-01]. Dostupné z URL: <https://basegroup.ru/community/articles/description>.

[17] *Random Forest*. RapidMiner Documentation [online]. [cit. 2017-04-07]. Dostupné z URL: <http://docs.rapidminer.com/studio/operators/modeling/predictive/trees/parallel_random_forest.html>.

[18] *K-nearest-neighbor*. RapidMiner Documentation [online]. [cit. 2017-04-07]. Dostupné z URL: <http://docs.rapidminer.com/studio/operators/modeling/predictive/lazy/k_nn.html>.

[19] TSARKOV, S. *Algoritmus nejbližšího souseda. BASEGROUP LABS* [online]. [cit. 2016-04-01]. Dostupné z URL: <https://basegroup.ru/community/articles/knn>.

[20] *Support Vector Machnine*. RapidMiner Documentation [online]. [cit. 2017-04-07]. Dostupné z URL: <http://docs.rapidminer.com/studio/operators/modeling/predictive/support_vector_machines/support_vector_machine.html>.

[21] *Naive Bayes Classifier*. RapidMiner Documentation [online]. [cit. 2017-04-07]. Dostupné z URL: <http://docs.rapidminer.com/studio/operators/modeling/predictive/bayesian/naive_bayes.html>.

[22] BAŽENOV, D. *Naivní bayesovský klasifikátor* [online]. [cit. 2016-04-01]. Dostupné z URL: <http://bazhenov.me/blog/2012/06/11/naive-bayes.html>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

GPU	Grafická procesorová jednotka - Graphic Processing Unit
ReLU	Rectified Linear Unit
k-NN	k-nejbližších sousedů - k-nearest-neighbor
SVM	Metoda podpůrných vektorů - Support Vector Machnies
NBC	Naivní bayesovský klasifikátor - Naive Bayes Classifier
CPU	Centrální procesorová jednotka - Central Processing Unit
CSV	Hodnoty oddělené čárkami - Comma-Separated Values

SEZNAM PŘÍLOH

A OBSAH PRILOŽENÉHO CD

bakalarskaprace.pdf elektronická verze práce

zdrojové kódy složka se zdrojovými kódy v jazyce Java a Python

- **serverClass.py**
- **server.py**
- **client.java**

váhy složka s použitými vahami

- **vgg16-weights.h5**
- **vlastni10-100.hdf5**

textové soubory složka s textovými soubory

- **classes-imagenet.txt**
- **classes-vlastni.txt**

instalator složka s navrženým instalátorem